Physics Engines in Mobile User Interfaces

Lisett Edström
Magnus Winter

Master's Thesis

Department of Design Sciences
Lund University
ISRN: LUTMDN/TMAT-5139-SE

EAT 2010

1 Abstract

Mobile devices, like mobile phones, are becoming more and more powerful with screen sizes and resolution increasing. With this development in computational power and improvements in screens, 3D Graphical User Interfaces are becoming more popular and the desire for dynamical and fun interaction with the devices grows.
One way of controlling and manipulating moving and colliding objects in applications in the user interface, long used in gaming, is to use a *physics engine*. The use of a physics engine in a user interface is not yet widely spread, but will probably become more and more common as companies need to find ways to differentiate themselves from each other.

This Master's Thesis report describes our investigation of how to improve the usability and user experience in a mobile user interface by utilizing a physics engine.

Mobila enheter, såsom mobiltelefoner, blir kraftfullare och skärmarnas upplösning och storlek växer. Med den nya datorkraft som enheterna besitter tillsammans med de bättre skärmarna har 3D-gränssnitt blivit populärare. Viljan att tillverka ett grafiskt användargränssnitt som är dynamiskt och roligt att använda blir större och större.
Ett sätt att kontrollera och manipulera rörliga och kolliderande objekt i applikationer i användargränssnitt är att använda en fysikmotor. Dessa datorprogram har länge använts inom spelindustrin, men har hittills inte utnyttjats i särskilt stor utsträckning i grafiska gränssnitt. Detta lär dock bli vanligare och vanligare i och med att företag behöver "sticka ut".

Denna examensarbetesrapport beskriver vår utredning kring hur man kan förbättra användbarheteten och användarnas upplevelse i ett mobilt grafiskt gränssnitt med hjälp av en fysikmotor.

Keywords: Mobile phone, User Interface, GUI, UI, physics engine, Bullet Physics.
Nyckelord: Mobiltelefon, gränssnitt, fysikmotor.

2 Preface

This Master's Thesis report describes the work done by us during the spring semester of 2010. The work was carried out at the Malmö office of The Astonishing Tribe, TAT and supervised by the Department of Design Sciences, LTH, Lunds University.

We would like to thank the following people without whom this Master's Thesis would not have been possible:

Dan Gärdenfors and Marcus Ericsson, our supervisors at TAT for taking us on.

Joakim Eriksson, our supervisor at LTH.

Mattias Wallengård, who put us in contact with TAT.

Michael Winberg and Karl-Anders Johansson for invaluable help with the programming.

The employees and Master's Thesis workers on TAT for ideas and good company.

Lisett Edström   Magnus Winter

Contents

3 Introduction

3.1 Goal of the Thesis

The Goal of the Master's Thesis was to investigate the possibilities, advantages and disadvantages of using a physics engine in a UI (user interface) in a mobile device. During our work we have taken a number of restraining turns to narrow the work load down to a reasonable level. Finally we decided to concentrate our work on investigating physics engines in UIs from an end-user point of view, i.e. how the usability and/or the user experience could be improved.

We chose to put aside investigations concerning how to write general and transparent code, which would be an investigation from a programmers or developers point of view instead of the end users. Since the Bullet physics engine is written a bit too general for our purposes (see Section 4.3.1 on page 8) some kind of API (*application programming interface*) would be needed or otherwise a new, more UI-centered, physics engine should be developed. We also chose to focus on using a three-dimensional space as our simulated world since the objects would then be able to both look and behave more realistically and to minimize limitations in movement and appearances due to the small screens on the devices.

The physics engine could also be used in other kinds of user interfaces than GUIs, e.g. input UIs.

Because of TATs policy regarding company secrecy we are not allowed to elaborate about these other possible continuations of our work in this report.

The reason that we chose to investigate the end user experience instead of other paths was that both we and our supervisors at TAT felt that without a clear motive on why to use a physics engine in the end user case, the other possible routes would be superfluous.

3.2 TAT - The Astonishing Tribe

TAT is a Malmö based software and design company with offices in Gothenburg, Seoul and Chicago. TATs main occupancy is to create user interfaces for mobile devices for both mobile phone manufacturers and mobile operators who wants to customize the interface in the phone.

The developed products strive towards being usable, on a high technical level with beautiful designs, something that is noted in their motto: “Design ♡ Technology”.

3.3 Background

3.3.1 Physics engines in general

A physics engine is a computer program which, using mathematical models of physical properties, gives virtual objects simulated characteristics to resemble real world behaviour. Examples of these properties are mass, velocity, friction, gravity etc. A physics engine is general, i.e it is not specific for each particular scenario.

A major function of the physics engine is collision detection. The collision detector handles both collisions between objects, deformation of objects (for soft bodies) and applied forces. There are several advanced algorithms for calculating the collisions efficiently

3.3.2 The use of physics engines

Physics engines have up until this now been used in various fields of calculations. The ones most similar to our research, since they are specialized in using the GUI (*graphical user interface*), are game engines that are used to make computer games feel real. Physics engines have however been used before advanced gaming was even born to predict everything from weapon projectiles to weather conditions. The first digital computer, the ENIAC, was used as a primitive physics engine to, among other things, compute projectile landing places. [1] $^{p.123}$

As mobile devices becomes more advanced when it comes to input possibilities, the need arises to find ways to process all the in-data combined instead of from one single input source at a time.

3.3.3 Physics engines in user interfaces

The standard user interface in a computer or mobile device has long been the so called *WIMP* (*Windows-Icons-Menu-Pointing*) device. The WIMP-style is what is used in most operating systems, word processors and web browsers both on computers and on mobile devices. With new input possibilities in mobile devices, such as gyrometers, accelerometers, camera and gesture sensitive touch screens, the WIMP-style loses ground and with 3D interfaces the windows and menus can be left out almost completely.

With new sources of input and more dynamical interfaces, the need to keep track of moving items increases. One way of controlling this could be to use a physics engine.

Some of today's uses of physics engine, or part of them, are described in Section 5 on page 16.

4 Theory

4.1 Usability and User Experience

The term *usability* is defined in the ISO 9241-11 standard. The definition deals with three aspects of usability as follows "the extent to which a product can be used by specified users to achieve specified goals with *effectiveness*, *efficiency* and *satisfaction* in a specified context of use.". [2]$^{p.4}$.

Apart from the ISO standard definition there are loads of other descriptions and definitions of usability, with smaller or larger differences. Tullis and Albert have, in *Measuring the user experience*, identified three common themes in most of these different definitions. These are [2]$^{p.4}$:

- A *user* is involved.
- That user is *doing* something.
- That user is *doing* something with a *product*, *system*, *or other thing*.

Since usability is so well defined, even if there are many differing definitions, it is convenient to introduce a wider term for everything that a user feels when using the product, system or other thing. This is often called the *user experience*, or *UX* for short. In our work we have used 'usability' for everything that increases the efficiency, effectiveness or understanding of the system and 'user experience' for things that the user feels is cool, good looking or gives more satisfaction even though the actual performance of the UI is not improved.

4.2 User interfaces

A *user interface* or *UI* is the way a user interacts with a machine, computer, mobile phone, a toaster or any other device. The user interface handles input and output of information to and from the device.

The main focus of the thesis work has been *graphical user interfaces*, *GUIs*, which gives graphical output on a screen. Other types of user interfaces that are used in mobile devices are, amongst others; gesture interfaces, voice user interfaces, touch user interfaces and so on.

4.3 Software tools

4.3.1 Bullet Physics 3D

Bullet Physics 3D is a physics engine released under ZLib licence [3][p.5]. The licence text reads as follows: [4]

> Copyright (c) 2003-2010 Erwin Coumans http://continuousphysics.com/Bullet/
>
> This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
>
> 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgement in the product documentation would be appreciated but is not required.
> 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
> 3. This notice may not be removed or altered from any source distribution.

The simulated world

The Bullet world is a space in three dimensions where it is possible to add objects and physical forces. There are different world classes used for different purposes. E.g when using only rigid bodies the `btDiscreteDynamicsWorld` is sufficient. If there is a desire to use soft bodies the `btSoftRigidDynamicsWorld` must be used instead. The inheritance graph for the world classes is shown in Figure 4.1

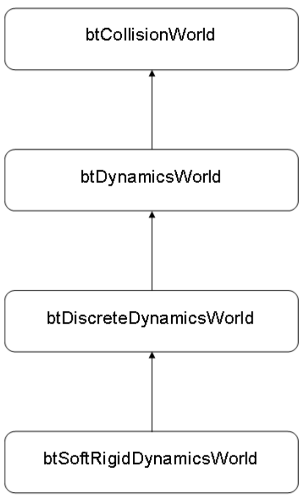

Figure 4.1: The world inheritance graph from Bullet

Rigid bodies

A rigid body is an object that occupies a three-dimensional space with geometrical properties, has a centre of mass and six degrees of freedom; three translational and three rotational. What defines the object as "rigid" is the property that the relative distance between two points remains constant over time regardless of any external forces applied to the object. This property is convenient when describing the position of an object since, as all internal distances are fixed, it is enough to know the position of one single point with three world coordinates (as long as the internal distances to the other points in the object are known). A convenient point to choose as reference point is e.g. the centre of mass or the origin of a local coordinate system fixed to the body. Every object has among other properties a mass, which can be infinite (in Bullet, this means that the object is static and is represented by a zero-mass), and a moment of inertia. It is possible to apply forces, impulses, velocities as well as torques and gravities that differs from the world gravity to the rigid body. Every rigid body has a collision shape which handles collision detection between objects. To create a rigid body in Bullet either a primitive shape (sphere, box, cone etc.) or a triangulated mesh can be used. Since the objects are not deformable the calculations needed every time step are only the transforms for the object as a whole (i.e translation and rotation).

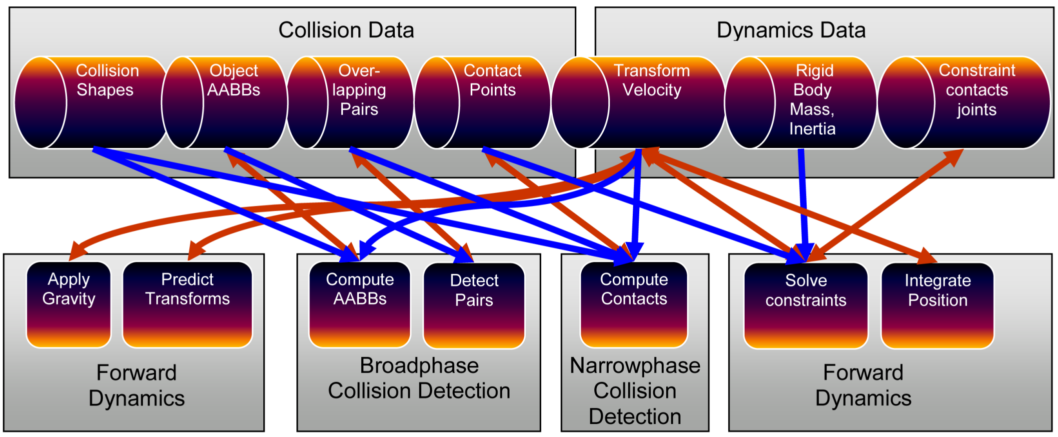

Figure 4.2: A pipeline from the Bullet user manual [3] $^{p.10}$ showing the most important data structures and computation stages in the Bullet Physics pipeline, executed from left to right.

Soft bodies

A soft body is an object which is deformable when external forces, e.g. air pressure, collisions or impulses, are exerted on it. That the object is deformable means that relative distances between internal points of the object are not fixed but rather change over time depending on external factors. Generally the body is expected to retain its shape to some degree and it is possible to control that degree by manipulating the physical properties of the object. Exceptions from this, i.e. bodies which do not retain their shapes whatsoever, are fluids. Since the internal relative distances are variable it is not possible to describe the whole object using a single reference point and a single world transform, rather all the nodes have to be specified in world coordinates and recalculated each frame. All objects are represented and constructed from triangulated meshes, and external forces can be applied either to individual nodes or to the object as a whole. It is possible to make specific nodes immovable or to attach them to a rigid body.

Collision shapes

Collision shapes are used to calculate collisions between objects. Unlike graphical meshes, a collision shape does not have to be composed of triangles but can be a primitive shape, such as a box or a sphere, to improve performance since all primitives have their own specialized and efficient algorithms for collision detection. A collision shape does not have its own world position but is attached to collision objects, rigid bodies or soft bodies. To save memory it is possible for the same collision shape to be used in several different rigid or soft bodies. In the cases where it is possible to use a primitive shape as opposed to one composed of a triangulated mesh, i.e. when the shape is right, it is preferable not only because it will improve performance, but also since it in some cases will result in a more accurate behaviour, e.g. when using round shapes like spheres or cylinders they will roll much smoother than the approximated polyhedral counterpart would.

Collision detection

Collision detection is one of the major parts of the physics engine. There are several different algorithms for detecting collisions between objects, some general and some specific for certain situations. To achieve better performance Bullet divides the collision detection into two different phases called *broad phase* and *narrow phase*.
The broad phase makes some approximations to quickly discard objects that are located so far apart that it is easy to determine that a collision has not occurred. All the remaining objects, i.e. the objects that might have collided with one another, are passed on to the narrow phase where it is calculated exactly which of these objects that were actually colliding. Additionally, for each of these collisions, an exact point of collision and the collision normal will be calculated.

In the broad phase the collision detection algorithm uses simplified shapes, such as e.g. spheres or boxes, instead of the actual shapes of the objects to make a quick and rough estimation of possible collisions. This means there will be fewer objects to consider in the more thorough, and therefore slower, narrow phase. These shapes are often referred to as *bounding volumes*. There are some primitive bounding volumes that are commonly used due to their simplicity. Properties that are desired for a geometrical shape to act as an effective bounding volume include: ([5] $^{p.76}$)

- Inexpensive computing
- Limited memory usage
- Easily rotating and transforming
- Inexpensive intersection testing
- Tight fitting

Which shape to use is a trade-off between how hard the bounding volume is to recompute and how expensive the intersection tests are to perform. The different bounding volumes are visualized in Figure 4.3. The simplest of these bounding volumes is the sphere. Since a sphere is rotationally invariable, only the translational transform needs to be recalculated as the contained object moves in the simulated world. To check whether two spheres have collided, calculate the sum of the radii of the spheres and check whether the distance between the centres

is less than this sum [6]$^{p.391}$. If so, the objects might have collided and are passed on to the narrow phase. This test has the advantage of being very simple, but unfortunately often results in too many objects being passed on to the narrow phase, since the bounding volume tends to be much larger than the actual object it encloses.

Another shape that is often used is the *axis aligned bounding box*, which is a box surrounding the object, constantly aligned to the world coordinate axis. This unfortunately means that the bounding box has to be recalculated every time the enclosed object rotates. To determine whether two objects are colliding the bounding volumes are first projected into one-dimensional space for every world basis axis and then checked to see if they are overlapping. Overlapping on all three axis means the bounding volumes are intersecting and that the objects should be passed on to the narrow phase.
An axis aligned box tends to fit the object better than a sphere resulting in less false collisions being sent to the narrow phase. However, some objects will still end up with bounding volumes much larger than the actual sizes of the objects, e.g. a thin, long object rotated 45 degrees in the x- and y-axis.

To make the bounding volume fit the object better one approach is to use an object aligned box instead of a bounding volume with axis fixed in world coordinates. The object aligned box has the same orientation as the enclosed object. This means that the orientation of the bounding volume must be recalculated whenever the object rotates. However, the volume of the bounding volume will be constant and therefore only needs to be calculated once, in contrast to the axis aligned box where the volume needs to be recalculated every time the object rotates. The drawback is that it is harder to have a general formula for calculating an object bounding box.

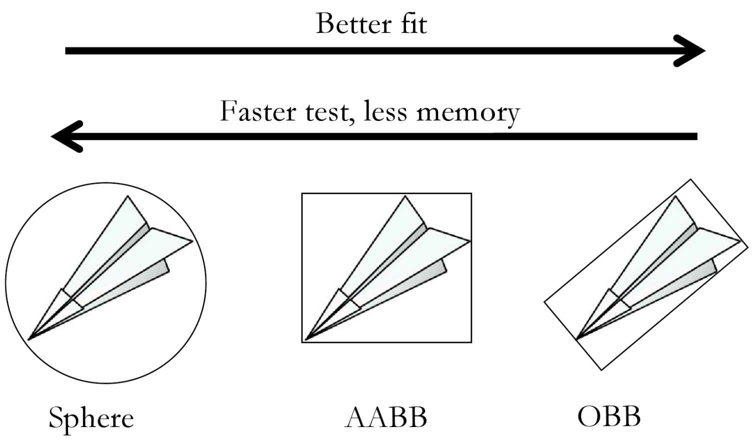

Figure 4.3: An object enclosed in different bounding volumes, [5] $^{p.77}$

Bullet has got a couple of different broad phase algorithms to choose from, such as *Sweep and Prune*, also known as *Sort and Sweep*, and *Dynamic Axis Aligned Bounding Box Tree*, more commonly called *Dynamic AABB Tree*.

Sweep and Prune

The algorithm starts by projecting the objects onto the x- y- and z-axis, resulting in a one-dimensional collision interval $[b_i, e_i]$ representation of the object (see Figure 4.4) with one point, b_i, marking the beginning and one point, e_i, marking the end of the interval. These two markers are inserted in a list which is then sorted in ascending order. This list is traversed from the beginning to the end and every time a b_i marker is found, object i is added to a list of active objects and whenever a e_i marker is found the object is removed from the list of active objects. Since it is only possible for a collision to have occurred if the objects are overlapping, there is only need to check collisions between objects that are in the active list simultaneously. If the intervals are overlapping for all of the three projected collision intervals, the objects concerned are passed on to the narrow phase. To improve performance for the calculation of the collisions, the algorithm makes use of the fact that frames are recalculated rather frequently, and that there are therefore good chances that objects have not moved much since the last frame. This temporal coherence is exploited in the updating of the lists. [7]

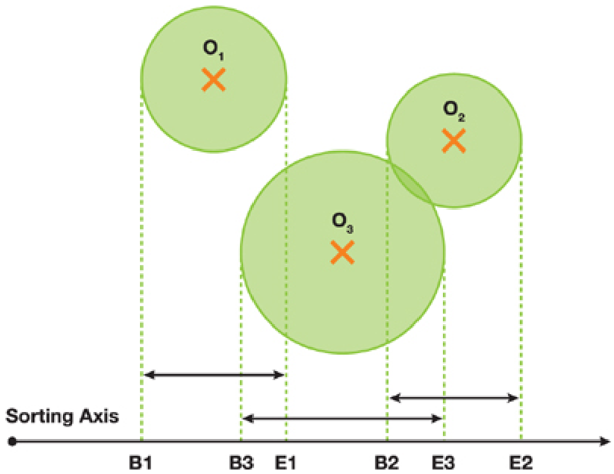

Figure 4.4: The Sweep and Prune Algorithm for three objects, image taken from $[7]^{sec.\ 32.1.1}$

Dynamic Axis Aligned Bounding Box Tree

This algorithm, Dynamic Axis Aligned Bounding Box Tree, is the one used by Bullet as the default algorithm for the broad phase of the collision detection.
All objects in the world are surrounded by bounding volumes, in this case axis aligned bounding boxes, which will make up the leaves of the hierarchical tree [8]. The objects are then grouped in small sets which in turn are surrounded by larger bounding volumes and inserted as parents to the single objects. These sets are then grouped into bigger sets and surrounded by even larger bounding volumes and then inserted in the tree. This continues recursively until all the objects are enclosed in a single bounding volume that will become the root of the tree, see Figure 4.5. The construction of the tree can be performed top-down, bottom-up or by insertion. The intersection testing is performed recursively between two nodes at a time and differs in its design depending on whether the nodes are leaves or internal nodes. A leaf is a bounding volume containing one single object whereas an internal node encloses a set of objects. For each pair of nodes, the bounding volumes for those nodes are tested for overlapping and only the nodes which return true for these tests are further traversed.

This results in three different cases:

1. Both nodes are leaves. The primitives of the nodes are tested for intersection and the result is returned
2. One node is a leaf and the other is an internal node. In this case the primitive of the leaf node is tested for intersection with each of the children of the internal node
3. Both nodes are internal nodes. When both nodes are internal, the node with the smallest volume is tested for intersection with each of the children of the node with the larger volume

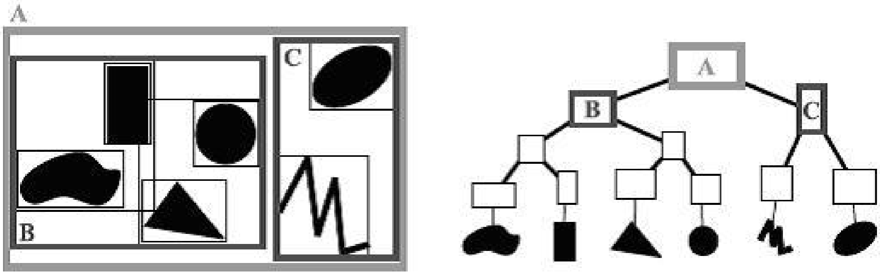

Figure 4.5: A hierarchical tree of bounding volumes, with rectangles used as bounding volumes, image taken from [9]

In the narrow phase calculations are performed to determine exactly which objects are colliding with each other in the simulation. To optimize performance, Bullet has specific algorithms in the narrow phase of the collision detection depending on what the types of the objects to be checked are (e.g. box-box, box-sphere, convex hull-sphere, convex hull-convex hull etc.). In order to determine which algorithm to use for a pair of objects, Bullet uses something called a *collision dispatcher*. The collision dispatcher takes two objects, determines their types and executes the correct collision algorithm for computing contact points.

It is possible to decide that only certain objects are allowed to collide with each other, e.g. in some games the character collides with walls and bounces back but when coming in contact with a power item the character should only receive additional powers, not be averted from the former trajectory.

Constraints

Bullet supports six different constraints. Most of them can be derived from the Generic Six Degree of Freedom constraint, however, lining them up will give an understanding of the possibilities of connecting objects in Bullet, hence we chose to describe them all.

Point to Point Constraint - is represented by the `btPoint2PointConstraint` class. It can be described as a ball point joint between two bodies.

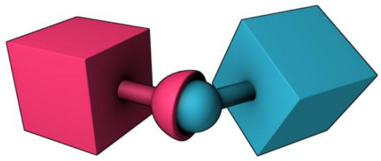

Figure 4.6: A point to point constraint, image taken from the Bullet user manual [3]$^{p.26}$

Hinge Constraint - is represented by the `btHingeConstraint` class. The Hinge constraint connects two bodies and lock them in all rotational axis but one, the hinge axis. The hinge constraint can for example be used to simulate doors or wheels that rotates around the axle tree.

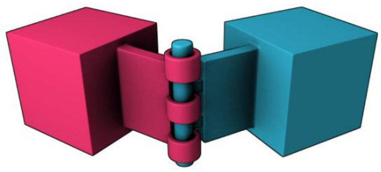

Figure 4.7: A sketch of how the hinge constraint can be visualized, image taken from Bullet user manual [3]$^{p.26}$

Slider Constraint - is represented by the `btSliderConstraint`. The slider constraint allows the connected bodies to rotate and translate round the slider axis.

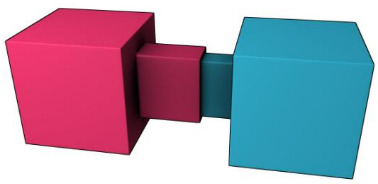

Figure 4.8: An image showing the slider constraint between two bodies. The possibility to let the object rotate round the slider axis is also available. Image taken from Bullet User Manual [3]$^{p.27}$
.

Cone Twist Constraint - is represented by the `btConeTwistConstraint`. The Cone Twist Constraint is mainly used when creating rag dolls for the connecting of limbs.

Generic Six Degree of Freedom Constraint - is represented by the `btGeneric6DofConstraint`. This constraint is a very general constraint which makes it possible to connect two bodies and allow, disallow or limit them to move in all translational and rotational axis.

Spring Constraint - is represented by the `btGeneric6DofSpringConstraint`. The spring constraint simulates a spring connected to one or between two bodies. The stiffness and damping, among other properties, of the spring can be set as well as the number of axis in which the objects are allowed to move and/or rotate. The constraint is a specialization of the `btGeneric6DofConstraint`.

4.3.2 TAT Cascades

TAT Cascades is a UI framework for creating and customizing user interfaces. The Cascades framework itself is written in C. The programmer can decide whether to use a static XML model or a more advanced Data Service written in C. Cascades is fast and efficient when running on devices and it is easy to try out different UI concepts since the compile time is quick.

4.3.3 TAT Kastor

TAT Kastor is a platform for rendering and creating UIs in embedded device environments. Kastor is, like Cascades and TML/TAT Motion Lab, developed internally on TAT.

4.3.4 TML and TAT Motion Lab

TML is an XML-like language developed by employees at The Astonishing Tribe. TAT Motion Lab is an XML develop environment for TAT Cascades which supports both text-based and visual programming. TAT Motion Lab works as an integrated production environment with TAT Cascades as well as a prototyping environment.

5 Related work and state of the art - Physics engines in user interfaces today

5.1 BumpTop

BumpTop is a graphical user interface, primarily developed for PC tablets and palmtops, based on the desktop metaphor (also known as a desktop ”environment”). It is designed to resemble a traditional, physical desk, see Figure 5.1, in contrast to the two-dimensional representation commonly used in computers today. BumpTop was originally created by Anand Agarawala as his Master’s Thesis at the University of Toronto, with the 1.0 version released on April 8, 2009. Since then, newer versions have been released, currently supporting Windows XP/Vista/7 and Mac OS X.

BumpTop simulates a workspace in three dimensions with a virtual desk and four surrounding walls. Objects, such as documents, photos, icons and short cuts are described as three-dimensional boxes with physical properties corresponding to properties experienced in the real world. This provides the user with the possibility to handle objects much as if they were real physical objects, e.g. pile them on top of each other, hang them on walls or toss them against walls or other objects to make them bounce. Newer versions of BumpTop also has support for multi touch input.

During the work with this Master’s Thesis BumpTop announced that they had been bought by Google. [10] Because of this the program is no longer available for free download, but not to wild a guess is that Google will use BumpTop for something in the future, maybe for the Chrome OS or Android OS.

Figure 5.1: BumpTop screen dump on a computer running Windows 7.

A selection of features include: [11]

- Stack objects in piles with advanced search and sort options to easily handle and organize the piles in a desired fashion, e.g. toss items at a pile, pop them into a grid or fan the pile out to view the content

- Pin sticky notes, favourite documents or commonly used short cuts to the walls. An additional feature is that files more frequently used will increase in size and weight to ease the access
- It is possible to have sociable icons, e.g. twitter or facebook, and interact directly with these, e.g. by to tossing a photo to a facebook widget to share it with your social network or by creating a photo frame on your wall to watch photo feeds from friends.

5.2 Physics engines in TAT UIs

On TAT, the notion of using physics engines to enhance the user experience and usability of user interfaces has been around for a while and some previous, however not extensive, efforts to implement applications with physics engines have been made. We interviewed one of the employees mainly responsible for these attempts and the full results of that interview can be viewed in Appendix B.

5.3 The iPod, iPhone and other Apple devices

In Apples mobile devices, such as the iPod or the iPhone, some elements from a physics engine are used to spring back lists when the top or bottom is reached. The scrolling speed is dependant on how fast the sweep was and some of the objects have some form of inertia. Apple are not keen on releasing information about their systems so the actual implementations of these functions have not been studied to any higher extent than by using the devices.

5.4 Physics engine enabled gestures in Windows Mobile 6.5

For the Windows Mobile 6.5 operating system, a physics engine API is available to interpret gestures as input [12]. This is done by checking for patterns in the input.

5.5 Physics engines in mobile games

Games developed for mobile devices have been a growing market for a while, and with the Apple Appstore and Android Market the industry has become even more extensive. The simplicity of creating applications and games and putting them up for sale on a virtual marketplace has resulted in that both companies and individuals with a programming interest have the possibilities to spread their applications world wide.

Especially games for mobile devices have been improved greatly during the last years. Not too long ago, Snake and Tetris was about what was possible to play on a mobile phone, but with better phones the games got better visual appearance and the next obvious step was then to attach a physics engine to your game (The fact that the Tetris blocks falls downwards does not really count).

6 Methodology

The Master's Thesis work was, during the time period that the work was carried out, decided to be focused on the possibilities and improvements of using a physics engine in user interfaces on mobile devices from an end users perspective. A large part of the work done has been discussions with employees at TAT concerning physics engines, both in general and more specifically concerning UIs in mobile devices.

We have done quite a lot of programming experiments with the physics engine connected to Cascades, trying out the different constraints (See Section 4.3.1 on page 14). We have also constructed some user scenarios including menu systems, transitions, lists and an application which uses the full 3D shape of a movable object to display information. The main focus has been to test physics engine concepts in combination with Cascades and therefore the visual appearances have been set aside. All objects have had a texture and we have used more complex shapes, e.g. triangulated meshes of a monkey head, to make sure that the physics engine worked correctly.

Some screen shots from our tests can be viewed in the results Section 8.

To explore our concepts and to receive new ideas and thoughts we organized a workshop with both designers and programmers attending. The structure of the workshop can be viewed in Section 7 on page 20 and the results in the Appendix A on page 35. The conclusions from the workshop are summarized in Section 9.2 on page 31.

During the work with the master's thesis a *NABC - Value proposition* was written. The value proposition described the needs, approach, benefits and competition of the developed product. The value proposition was created iteratively, i.e. it changed during the work as we came up with new ideas or ran into problems. The idea with the value proposition was to keep track of what was important during development and was additionally used to collect the information to be pitched to the customer. Since our end customer changed during the work our first value proposition was split into two versions, one that addressed the user of the phone and one that turned towards developing companies.

6.1 Existing application with animations

One of the tasks we decided to take on was to recreate an existing application which uses pre recorded animations to give a bouncy feel to the objects. We chose to recreate the weather application from TAT Home, a video of this can be seen at

`http://www.youtube.com/watch?v=bOGmnnGpoqg` at 2m53s [Accessed on May-5-2010].

Figure 6.1: The weather application from TAT Home in a lying state

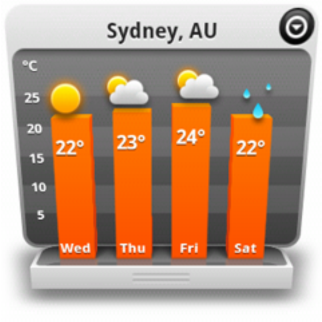

Figure 6.2: The weather application from TAT Home in a standing state

Our version of this application lacked the real textures and the meshes used for the animated application could not be used because of them being cavernous. However, we succeeded to recreate the feel of the application using the physics engine. To control the objects we used some short cuts such as giving the panel closest to the ground (the one that should snap) a higher mass than the other. This was needed since the hinge connecting the panels otherwise would have given the whole body a force big enough to swing back to the other state. To change the masses of objects (or applying constraints which would also have been an option) is not hard to do in Bullet. However, some extra work would not have been needed if the animation had been recorded instead.

7 Workshops and Interviews

7.1 User-case workshop

Workshop the 26 of march 2010
The workshop was carried out on the sixth floor at the Malmö office of TAT and had six participants. The goal of the workshop was to get new ideas for when it is desirable to use a physics engine in a mobile UI to increase the usability or the user experience.

The workshop was started by a very short introduction to basic physics and an explanation of what Bullet Physics can and can not do. The participants were however encouraged to brainstorm ideas even if Bullet could not handle the physics behind the idea. E.g. magnets could be a part of the idea although Bullet does not have exclusive support for representing magnetism.

The brainstorming part of the workshop was split into two parts. The first one was carried out individually and the attendants were to draw three cards by random; one with a UI scenario, one was an input and the last one was a constraint or a change in the physical properties. All cards are shown in Table 7.1

7.1.1 The card part

Scenario	Input	Constraint/Physical property change
Contact list	Press	Spring
Music Player	Drag	Hinge
Screen Saver	Drag & Release	Changed mass
Menu system	Double Tap	Deformation
Web Browser	Long Tap	Changed Gravity
	Shake	Impulse
	Tilt	

Table 7.1: The cards used in the workshop

The applications chosen were based both on the frequency of use and the fact that they differ in appearance and the way that they are used.

The different inputs were picked because they represent some standard inputs on a smart phone device on the market today; touch input and accelerometer.

The constraints and physical property changes were meant to be clear for all the participants, even without experience in physics and maths.

7.1.2 The free part

In the free brainstorming part of the workshop, the participants were asked to come up with ideas where a physics engine would improve the usability or user experience in a mobile UI. The results were not as profitable as the card part, however we got some nice advices along the way. All results from the free part of the workshop can be studied in Section A.2 on page 51.

8 Results

In our Master's Thesis work our goal was to investigate the possibilities, advantages and disadvantages of using physics engines in user interfaces on mobile devices. To accomplish this, our first task was to get to know the tools used on TAT, TAT Cascades (see Section 4.3.2) and TAT Kastor (see Section 4.3.3) and integrate these with Bullet Physics 3D, the physics engine we chose to work with during this thesis. It took us some time to perform the integration since we both had to keep track of, and influence, the objects in two different worlds simultaneously (Bullets and Cascades'/Kastors) and the conversion of these objects between worlds. We managed to get this to work smoothly and were then able to create arbitrary meshes and textures in the TML code, use them to construct rigid bodies in Bullet, add constraints, apply forces on objects and finally render them using Kastor.

We visualized our applications on a computer, instead of trying to run it directly on a device, which meant we had to find a way to simulate input. We chose to solve it by letting the mouse cursor represent touch input. This was easy and worked perfectly when we just needed movement along the x- and y-axis but took some adjustment when we wanted to be able to move objects along the z-axis, an issue which would also have arisen when running the applications on a device. This was solved by using the movement along the y-axis and interpreting the movement of the cursor upwards as, depending on scenario, movement of the objects upwards and/or backwards, into the screen and moving the cursor downwards meant moving objects downwards and/or closer, towards the front of the screen.

The weather application

One part of the research was to take an existing application, which was done with recorded animations, and try to recreate it by using the physics engine instead. The application in question in this case was a weather application from TAT Home described in Section 6.1 on page 19. We decided to simulate each one of the bars, the back panel and the bottom panel as separate objects, connected with springs and hinges. This meant that all the bars would be allowed to move, swing and shake independently from each other, within limited intervals. Additionally both the back panel and the front panel was set to oscillate before settling in the one of stable states (see Figure 6.1 on page 19 and Figure 6.2 on page 19). The reason for deciding to use these features were that we wanted the application to feel more alive and act more like users would expect objects to behave in the real world, i.e. when flipped and moved they continue to oscillate before settling.
Creating the bodies of the objects did not present any problems, however combining them with constraints (hinges and springs) was a bit harder. A lot of the problems originated from objects trying to occupy the same space in the world, which resulted in frantic behaviour of the objects, e.g. extreme forces applied to them, wild shakes etc. One of the reasons of the objects ending up in the same place was that Bullet suffers from a severe lack of documentation which became extremely apparent when creating a constraint between two bodies and trying to decide its location. It was hard to determine whether Bullet wanted to have the absolute or relative coordinates for the bodies, or perhaps absolute coordinates for one object and relative for the other.

The user had the possibility to directly influence and decide the velocity of the transitions between states. This because we connected the touch input to the force applied on the objects and hence controlled the speed of the movements. It was also possible for the user to start flipping, change direction and simply hold the combined body in between states to get sneak peaks of

the information displayed on the other side. An additional consequence of letting the input directly control the forces applied to the objects was that when moved slowly, the panels and bars would hardly shake at all whereas a quick flip would result in extensive oscillations of the body.

The menu system

The workshop we arranged at TAT gave us inspiration for new possible scenarios (for the result of the workshop, see Appendix A on page 35), and we decided to try and implement one of them to see how it would work in a real application. The scenario we chose was "Radial Madness" (see Figure A.11 on page 46). The concept here was to start out with a single button representing the root of the menu system, see Figure 8.1. When clicked, the sub menus would appear surrounding the menu button, attached to this with springs, see Figure 8.2. If clicked again the sub menus would disappear. The same concept was used for the sub menus; if there were sub menus to these (e.g. if "Games" was a sub menu to the root button, "Snake" and "Tetris" could be sub menus to "Games") when clicked, their sub menus would appear surrounding it, attached with springs, see Figure 8.3, and another click would make them disappear. If a button without sub menus was clicked it would mean that the end of the tree had been reached and that an application should be opened. This transition was done by releasing all springs and apply random forces to the visualized objects to make them start moving out of the screen. The reason for choosing random forces was that we wished to give our application a unique appearance every time it was run, something that would not have been possible to do using pre recorded animations. This system worked really well and by applying random forces to the objects, which gave the them springy motions and made the system behave like it would have done had it been a device containing real physical objects on real physical springs, we simulated how it could look if we were to let input from the accelerometer (shakes and tilts) represent forces in the world.

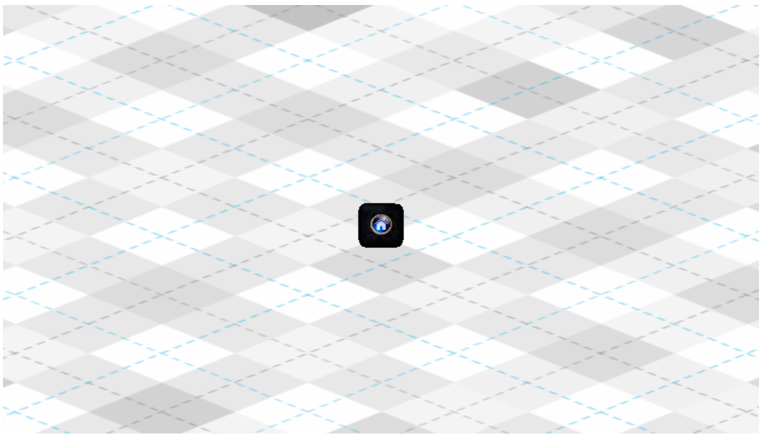

Figure 8.1: A single menu button, representing the root of the menu system tree

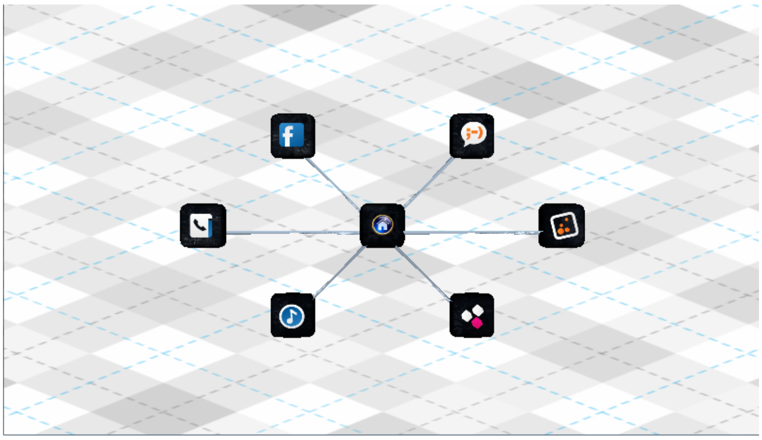

Figure 8.2: The sub tree to the menu button

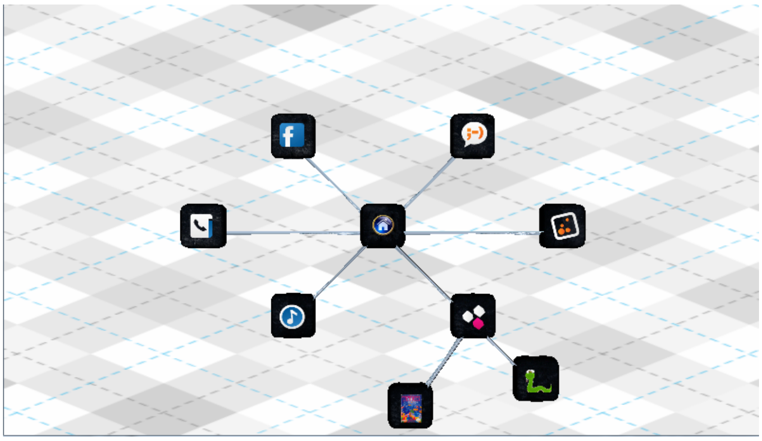

Figure 8.3: The next level of sub trees

The contact list

We also had a notion of how we could use the physics engine to visualize different properties of objects in lists, e.g. ranking of songs in a music player, the amount of information in the different objects in a file system, recency and frequency of use of specific contacts in a contact list etc.

- *Ranking of songs in music player*
  Our idea for this scenario was to use the ranking system to determine how light/heavy the objects should be. A quick way to sort the list would then be to give objects with few stars a higher gravity compared to the objects with more stars, which would make the less popular objects end up at the bottom of the list while the favourite songs would end up at the top.

- *Information in files*
  One way to deal with this scenario was to use gravity or the size of the objects to visualize the amount of information in files, e.g. more information makes the file increase in size or gives it gravity towards the front of the screen while less or no information makes the object shrink or gravitate further into the screen, away from the user.

- *Recency and frequency in contact list*
  Here we also had the notion that it would be possible to use gravity or perhaps friction to visualize how frequently or recently specific contacts in the contact list had been accessed. A seldom used object would gravitate further into the screen, away from the user and a frequently used contact would end up at the front of the screen when the contact list is sorted.

Of these scenarios, we decided to further explore and implement the last one, the visualization of the contact list. We created simple objects representing the contacts and sub trees for these, consisting of objects containing information about the contact, e.g. phone number, email address and profile picture. The contacts were arranged in a three-dimensional list orientated so that the first object is closest to the screen and the following objects aligned vertically and horizontally but situated further into the screen. The objects are connected with springs to give the list a softer and more lively behaviour. When a contact is clicked, the nodes with information appears distributed above the contact, attached with springs (the same principle as for the sub menus in the application visualizing the menu system). A distinct shake of the device from the user releases the springs between the contacts and makes them fly around. A new shake recreates the list, now sorted after the frequency of use, i.e. the contact most frequently used will end up closest to the screen, the next most frequently used contact will end up after this and so on. A different shake (e.g. one of the shakes could be horizontal and the other vertical) would result in the list being sorted after how recently, instead of how frequently, the contacts have been used when recreated. Since it was not optimal to shake the computer as input, we simply created two additional objects, which if clicked, one representing the wish to sort after frequency and the other after recency to simulate receiving input from the accelerometer.

Vizualisation of physical properties

After implementing these features we decided to continue working with the list and further develop the dynamical behaviour. We decided that instead of rearranging the order of the objects we were going to use another way of visualizing the frequency of use of the objects. The approach we chose was to apply gravity upwards, with the magnitude of the gravity determined by how often the objects had been used so that favourite objects would gravitate up high while objects rarely used would more or less stay in their original position. This meant that objects that were used frequently would be simulated higher up on the screen and therefore more easily viewed and accessible since there would not be as many other objects concealing them, see Figure 8.4 [1].

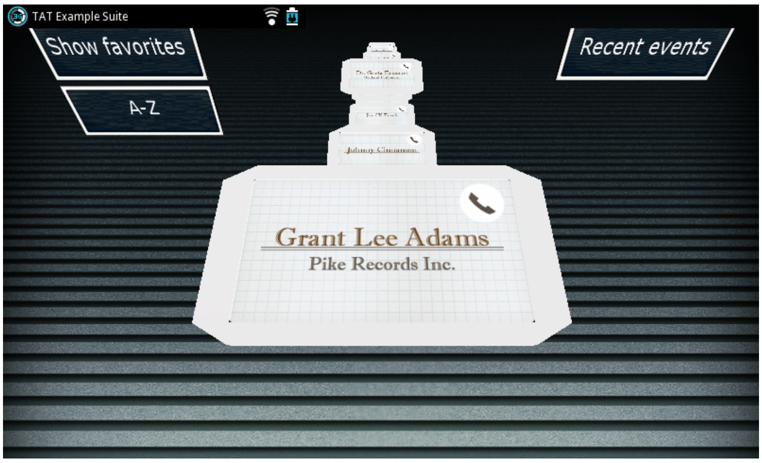

Figure 8.4: The objects have been given a gravity upwards with the magnitude of this determined by how frequently they have been used. Objects with high frequency of use are situated higher up on the screen whereas seldom used objects have more or less stayed in their original position.

Switching between states

We also identified another part of the list application where it was favourable to use a physics engine. We decided to have two different modes, one that showed all the contacts alphabetically and one that only listed a predetermined number of the contacts in the order of how frequently they had been used. The transition between these two modes performed in the following way:

- We started by removing all constraints so that the objects no longer were connected to each other.
- In "favourite" mode, objects that were no longer to be shown were given a gravity downwards and as they fell they started to fade (see Figure 8.5) until they after a while became totally invisible, see Figure .

[1] We chose to simulate the input on the computer by clicking the buttons "Show favourites", "A-Z" and "Recent events" so on the device these would not be present and the list would have filled the whole screen in portrait view

In “alphabetical” mode, the objects that were not displayed in the list of favourites slowly became visible, simulated further down than the other objects (see Figure 8.6, and were starting to gravitate upwards, see Figure .

- The objects that were to be displayed in the list were given destination coordinates depending on where in the list they should be visualized, i.e. the first item in the list (depending on mode, either the first one alphabetically or the one most frequently used) should always be simulated closest to the user and then the next object a fixed distance further into the screen and so one. The objects where then given gravity towards these destination points. To avoid the objects from getting in each others way, piling up or pushing each other in the wrong directions during these transitions, we simply told Bullet that the objects were not allowed to collide (by changing a collision flag).
- When all objects had reached their destinations they were once again connected with (invisible) springs making them form a new coherent list see Figure 8.7 and Figure 8.8.

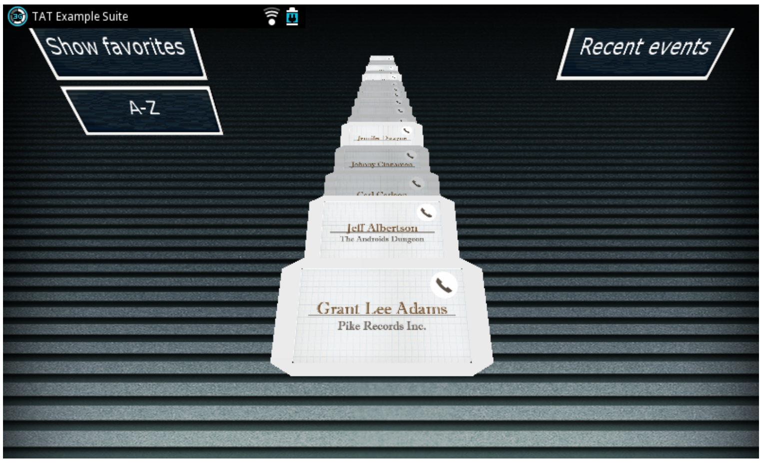

Figure 8.5: The user has just chosen to switch from alphabetical mode to favourite mode. The objects that are not to be shown in the list of favourite objects are starting to fade.

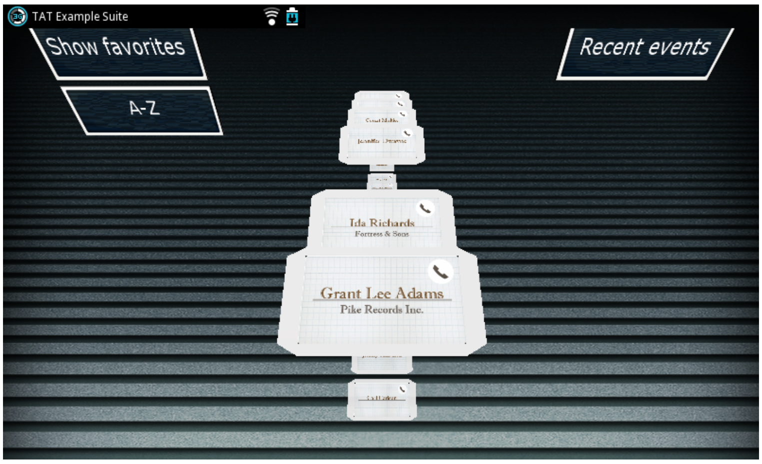

Figure 8.6: The user has just chosen to switch from favourite mode to alphabetical mode. The objects not shown in the list of favourites has just become visible and are starting to gravitate upwards to join the list.

Figure 8.7: The list displayed in alphabetical mode

Figure 8.8: The list displayed in favourite mode

The use of the physics engine in this case resulted in a transition that was constantly fully consistent with previous history of interaction with the objects, i.e. how often contacts had been used, a feature which had been if not impossible at least very hard to achieve with pre recorded animations (unless the number of contacts was extremely small).
This approach to the transitions between two states also had the advantage of the user always being able to dynamically follow the repositioning of objects instead of having the objects just magically appearing in a completely new order.

9 Discussion

9.1 Discussion about the usage of physics engines

Creating objects

When we had gotten the physics engine up and running together with the Cascades viewer we realized that we needed some way of sorting out the objects that were to be simulated in Bullet. We solved this by naming objects and parsing the names, however this is not a method that works well for larger UIs since the names get confusing after a while. However, for our purpose of testing out the possibilities of the integration between a physics engine and TATs software (or any graphical renderer for that matter) this method worked fine.

We noticed that we could easily play around with the objects when we wanted to, and as easily lock objects in different axis or planes. Attaching constraints was a bit tricky, but we managed to find out all the parameters by trial and error. As mentioned before, the documentation of Bullet is quite inadequate. Since Bullet sometimes gives strange forces when connecting objects the need to calm the objects down arose and this was most frequent solved by setting their velocity to zero after being attached to something else. Why these impulses are applied are quite a mystery to us, but since it is so easy to stop them we did not put much thought into it.

The weather application

The conclusion drawn after doing the replica of the weather application was that it might be superfluous to use the physics engine when only dealing with one object, even if the actual object was constructed by multiple ones. The work with connecting all the objects with hinges and springs in the correct places and to specialize the input to move the object in a good looking way as well as the snapping of the right panel to the ground resulted in the gain of using the physics engine, instead of the pre recorded animation used in TAT Home, being very low. The only thing that we could achieve that the pre recorded version could not was that the bouncing and oscillating of the panels and bars looked different each time we moved it, however these movements were quite small since we did not want objects to take too long to stabilize since the user would be annoyed by the wait.

The menu system and contact list

With the implemented list we could see that the physics engine gave a bigger gain than in the single object case since we could easily get all objects to move around at the same time when releasing the springs and when we wanted to connect them back again we could simply use the same algorithm that we used to position them from the beginning. The fact that the objects behave the same way every time while ordered and connected to each other, but never in the same way while floating around is great since the ordered state is the one where the user actually uses the list and the floating state gives the application a feeling of uniqueness every time it is used.

We felt that visualizing the frequency of use of the contacts by giving them gravity and destination points proportionally worked really well. The resulting positioning of the objects gave the

user a clear and lucid view of their popularity and made it easy to quickly distinguish and access the most commonly used contacts. Also the possibility for the user to continuously follow the movement of the objects instead of just ending up with them in a new state without being able to view the steps in between (as is often the case today when switching between different list views) increases the feeling of being completely in control of the situation and decreases the time it takes to readjust and find the desired contacts after they have been repositioned.

Rendering errors

Even when objects are not moving some rendering errors could sometimes be seen. This effect is noticed on sharp edges on objects and probably comes from rounding errors when going from floating point numbers to fixed point, or from errors within Bullet. Floating and fixed point numbers are two different ways of representing real numbers on computers. The floating point numbers are used in Bullet and have the possibility to represent a larger range of numbers since the decimal point can be moved. Fixed point numbers have a fixed number of decimals and a fixed number of integers and are used in Kastor.

2D versus 3D

Although we have focused on 3D UIs during the whole work, to use a physics engine in 2D is possible, but since the screen size was small the need for it seemed smaller than in the three dimensional case. We also felt that it would be more interesting to try to simulate the objects to look and behave as much as possible as physical objects in the real world would. It would also be quite easy to project the simulations from 3D to 2D just by locking objects from rotating and translating in one of the dimensions, which motivated us to concentrate our efforts and time in exploring three-dimensional applications.

The choice of physice engine

In our research we used Bullet as a physics engine since it is a very powerful and capable tool with efficient support for handling a lot of objects simultaneously and with optimized algorithms for calculating the behaviour of these objects due to the current forces present in the simulated world. However, for our purposes we found out that Bullet might be a bit over qualified, i.e. there are many parts of it that become superfluous for our needs and uses. Additionally, from out point of view it would be desirable to be able to smoothly and easily put restrictions on objects, which might not be physically correct but which would make the interface behave in the desired or intended way. The solution to these problems, or concerns, we concluded that either it is best to implement a new physics engine, customized for our needs, or to implement an API (*application programming interface*) that acts as an intermediary between Bullet and the programmer. When implementing a new physics engine it would probably be possible to use a lot of the algorithms from Bullet so that everything does not need to be created from scratch. And with customized physics engine it would also be possible to solve the problems with being able to put restrictions on the behaviour of the objects. However, if the need arises to use a lot of the features in Bullet, perhaps when creating more extensive applications than we did, it might be preferable to simply use the other solution and create an intermediate API.

Issues concerning screen sizes

One thing that became apparent during our work was that the size of the screen heavily limited the effects that were desirable when using a physics engine. The whole point of using a physics engine in the application was to make objects move and collide according to forces and impulses exerted on them and if there were no movements, the application would look exactly the same if the physics engine was removed. Unfortunately since objects on mobile phones need to be at least of a size where it was possible to see what they represented and where it was possible to press them, this put severe limitations on how much screen surface was available for moving them around. However, this is not so much of a problem when dealing with larger screens which are becoming more and more common, e.g. tablets such as the Apple iPad. There were still distinct effects from the physics engine on the screens we used, they would only have been more extensive had the screen surfaces been larger.

9.2 The workshop

The card part

The ideas created by the participants in the card part of the workshop can be seen in Section A.1 on page 35.

The card part was successful in the aspect of the amount of differing ideas received. That the randomized combinations were not always "natural" encouraged the participants to think outside the box which was part of the goal. There was a general aversion towards using the physics engine when the application was the web browser. This was anticipated since the main expectation of a web browser, especially on the small screens of mobile devices, is to display the content of a web page and not much more. We still decided to include the web browser as an application card since we wanted to investigate new possibilities of using a physics engine.

Some of the ideas from the workshop concerned transitions. The advantage of using a physics engine for transitional purposes from the participants point of view was that the user could be more in control of deciding for example the speed of the transition. There was also the notion that the physics engine would enhance the visual experience of the transition. This was suggested to be done with both cloth (shown in Figure: A.2) and a liquid (shown in Figure: A.14) to give the transition a more real life feeling.

One of the more popular applications to work with was the screen saver. Even though the actual usability was not improved the participants felt that there would be an improvement in the user experience giving the users the possibility to play around with objects without starting an actual game. There are a lot of situations where people holds their mobile device without the actual intention of doing anything productive with it. In the screen saver it does not matter if the behaviour of objects are out of control, here it can even be desirable. Workers at TAT have experienced, when showing new products, that users find enjoyment in intentionally trying to beat the system, i.e. getting the UI to do things that it is not designed for. An example from the workshop can be seen in Figure: A.5.

When studying the outcome there seems to be a tendency among the participants to be willing to use the physics engine in some applications whereas in others the participants seemed to find it hard to invent new ideas concerning the UI. E.g. no one came up with ideas that changed the appearance of the web browser but rather appeared determined to keep it in a familiar fashion.

There were neither any changes to the controls, nor to the transitions between different web browser windows etc.

The free part

The collected impression from the participants were that physics engines are mainly used for effects, such as springy lists, throwable objects, hinge-like transitions etc. Some of these effects could be done with pre recorded animations, however throwing a large number of objects around and allowing it to look real calls for some sort of physics handling.

The idea where the whole system becomes slower and heavier to indicate that the battery is running low would mean that the whole UI is connected to the physics engine, the big problem with that scenario is to make it in a way that does not annoy the user.

Even if the participants did not write it down, there were quite some ideas concerning the fun part of playing around with objects, e.g. in screen savers/idle screens. This does not call for the same amount of control of the objects, it does not matter if some boxes get thrown off the screen or get an unrealistic high velocity. In more non-playful parts of the UI the control of the objects must be stricter.

9.3 The interview

Even though we only did one actual interview, we got loads of ideas and comments from employees at TAT during our work. The collected impression, which also is acknowledged in the interview is that the use of physics engines are desirable, but that it is hard to find good user cases. Our interviewee confirmed our notions that controlling the physics engine is often so hard that it makes it impossible to use. The lack of algorithms to snap objects to their desired states is also a big problem. In user interfaces the gain from getting the realistic feeling is often less than the problems that the physics engine cause. Like we have established, the interviewee states that a good thing for the future would be to implement a physics engine that is specialized, which only simulates things as far as needed and when closing in on the final state of an object, other algorithms than real world simulation takes over.

10 Conclusions

During our work we found several cases where usage of a physics engine enhanced the user experience of the applications in the user interface. The use of a physics engine resulted in unique appearances and behaviour of the interface when objects were allowed to move around and/or collide, which would not have been possible to achieve by using pre recorded animations. It was also easy to deal with large amounts of objects, e.g. items in a list, without additional work, another advantage to using pre recorded animations. The user interface, and the objects in it, felt alive and it was possible to interact with them in a dynamical and satisfactory way. It was also easy to visualize different properties of objects, e.g. how frequently they had been used, in a way that was clear to the user. Since it was possible to move objects around dynamically it became a convenient way of switching between states that meant the user never had to lose focus of the objects, but could rather follow their movements as they were repositioned instead of just suddenly seeing a completely new view where objects were missing or appearing in a different order.

One conclusion we reached was that the optimum way to use the physics engine would be in combination with pre recorded animations to achieve the most satisfactory user interfaces. This would make it possible to combine the best qualities of each component and fully exploit their potentials and strengths and eliminating the weaknesses present in an interface using only one of them.

However, there is a lot of work involved when using a physics engine. Objects are sometimes going into resonance, ending up with huge velocities or getting stuck in walls and/or each other etc. In our case, using Bullet as a physics engine was handy since it was a complete, fully functional physics engine, but unfortunately this was also what caused us trouble. Bullet had too many features and was too physically correct for our needs. In our applications it was not always desired to have the objects behaving in a perfectly realistic way but rather in a way that resembled the real world but was adapted to ease the usage of the interface and to enhance the satisfaction and diminish the frustration of the user. It was hard to find the right parameters for objects and constraints, which meant we often ended up with objects going haywire. However, an easy solution to this problem could be to implement a physics engine designed and more suitable for the needs and uses of the applications in the user interface.

11 References

References

[1] R. Rojas and U. Hashagen, *The first computers: history and architectures*. MIT Press, 2000, ISBN: 0-262-18197-5.

[2] T. Tullis and B. Albert, *Measuring the User Experience*. Morgan Kaufmann Publishers, 2008, ISBN: 978-0-12-373558-4.

[3] E. Coumans, *Bullet 2.76 Physics SDK Manual*, http://bulletphysics.com/ftp/pub/test/physics/Bullet_User_Manual.pdf, 2010.

[4] Bullet license document. http://code.google.com/p/bullet/source/browse/trunk/BulletLicense.txt. [Accessed 28-April-2010] - Local saved copy is available.

[5] C. Ericson, *Real-Time Collision Detection*. Morgan Kaufmann Publishers, 2005, ISBN: 1-55860-732-3.

[6] M. DeLoura, *Game Programming Gems*. Charles River Media, inc., 2000, ISBN: 1-58450-049-2.

[7] H. Nguyen, *GPU Gems 3*. Pearson Education, 2005, ISBN: 0-321-51526-9.

[8] G. van der Bergen, "Efficient Collision Detection of Complex Deformable Models using AABB Trees," 1998, [Accessed 4-May-2010].

[9] Wikipedia. (2010) Bounding volume hierarchy — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Bounding_volume_hierarchy&oldid=354257931. [Accessed 4-May-2010].

[10] BumpTop. (2010) BumpTop - homepage — An important BumpTop announcement. http://eol.bumptop.com. [Accessed 3-May-2010] - Local saved copy is available.

[11] Wikipedia. (2010) BumpTop — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=BumpTop&oldid=358522528. [Accessed 28-April-2010].

[12] Microsoft Corporation. (2010) Using gestures in windows mobile 6.5. http://msdn.microsoft.com/en-us/library/ee220920%28v=MSDN.10%29.aspx. [Accessed 5-May-2010].

A Appendix: Outcome of the workshop

Here follows the ideas the participants of the workshop came up with. The images are recreated by us from sketches made by the participants. We chose to do this because we wanted to enhance the level of understanding of the scenarios. The participants were encouraged to make quick and simple sketches so that the time was spent inventing instead of drawing.

The analysis and conclusions taken from the workshop are described in Section 9.2 on page 31.

A.1 The card part

Menu system - Shake - Deformation

1. The icons in the menu system are organized in a grid.
2. The mobile phone is shook.
3. The icons fall down and pile up at the bottom of the screen.
4. Where the icons were, there is something that the user wishes to have quick access to.
   E.g. Received text messages, previously dialled numbers or the calendar.

Figure A.1: Menu system - Shake - Deformation

Menu system - Drag - Impulse

1. The menu is represented as a bar on the left hand side of the screen.
2. The menu receives an impulse from a drag touch input.
3. The menu is folded out so that it covers the whole screen in a smooth way, like a light curtain.

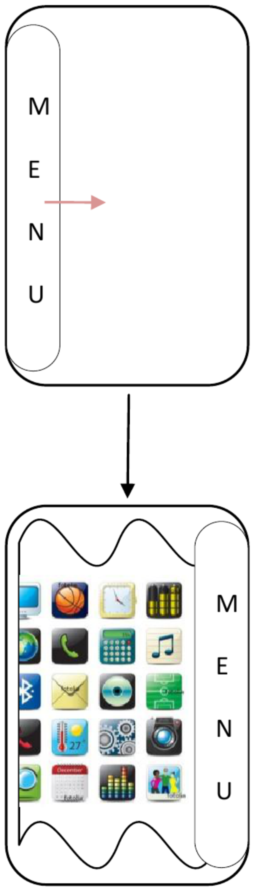

Figure A.2: Menu system - drag - impulse

Contact list - Press - Spring

1. A contact list is shown.
2. A press is made on one of the contacts.
3. The contact is shot out as if it were attached on springs, just like a cuckoo in a cuckoo clock.

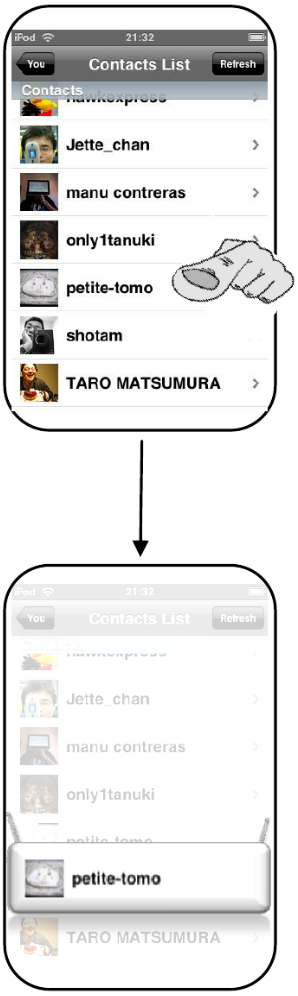

Figure A.3: Contact list - Press - Spring

Web browser - Press - Impulse

1. The web browser covers the whole screen.
2. A press input gives the browser an impulse so that it is pushed back into the screen and hence will shrink.

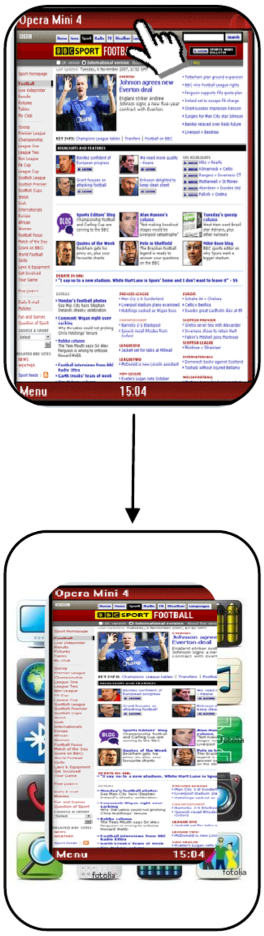

Figure A.4: Web browser - Press - Impulse

Screen saver - Drag & Release - Impulse

1. Spheres are floating around in a three dimensional space on the screen.
2. One of the spheres is grabbed and tossed.
3. The resulting collisions, velocities and forces are dependant on how fast and long the drag was.

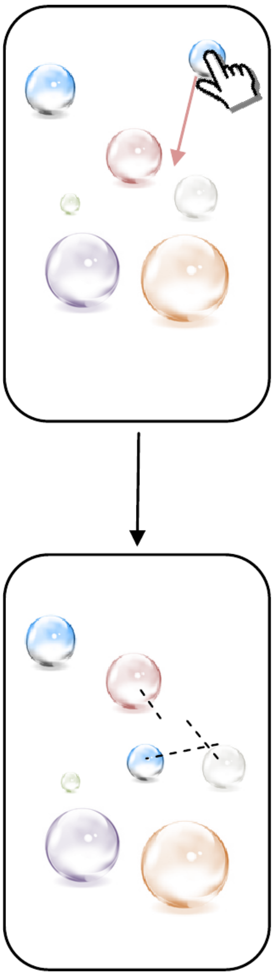

Figure A.5: Screen saver - Drag & Release - Impulse

Screen saver - Double Tap - Hinge

1. A rotating slide show of pictures attached by hinges is shown.
2. A double tap will result in changes of the rotation.

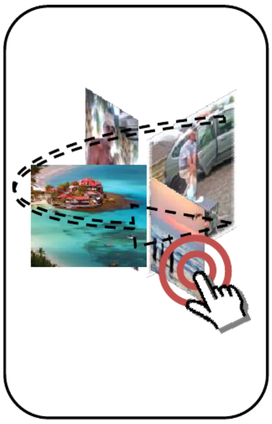

Figure A.6: Screen Saver - Double Tap - Hinge

Music player - Drag & Release - Spring

1. The music player, attached with a vertical hinge in the middle of the screen, is shown.
2. By dragging the whole screen the music player acts like a door.
3. On the other side of the "door" additional information about the current music (artist, track, etc.) is shown.
4. When released, the music player will snap back to its original position.

Figure A.7: Music Player - Drag & Release - Spring

Menu system - Long Tap - Changed mass

1. A menu system organized in a grid is shown.
2. A long tap on one of the icons is made.
3. The tapped icon will become heavier.
4. The longer the press, the heavier the icon becomes.
5. When the mobile phone is shook the grid is released and lighter (or heavier) icons slide away from the screen.

Web browser - Press - Deformation

The rendered web page is emulated as a membrane. This could be useful using a device with touch input on the backside.

Web browser - Shake - Impulse

The device is flicked sharply in either an upward or downward direction which results in a quick scroll to the top or bottom of the shown web page. The motion is similar to the one used when expanding a telescopic batons.

Screen saver - Long Tap - Spring

Attached particle systems to the position of the touch input when long tapping. On a short tap input the system emits new particles.

Web browser - Tilt - Hinge

1. The web page is attached to a hinge on one side of the screen.
2. When the device is tilted sideways the page swings which changes the scroll agility.

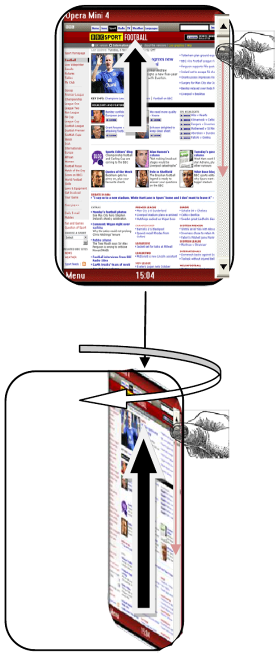

Figure A.8: Web Browser - Tilt - Hinge

Menu system - Double Tap - Changed Mass

Toggle icon masses with double tap gestures. Toggle between two discrete states: light and heavy. Use shake or tilt to sort.

Contact list - Tilt - Impulse

A quick tilt results in a scroll up or down by giving the list an impulse.

Music Player - Shake - Changed Gravity

1. A play list with items is shown. The gravity is normal (by Earth standard).
2. Gravity is set to 0.
3. Shaking the device results in a shuffle of the items.
4. Gravity is reset.

Menu System - Shake - Changed Mass

1. A menu system with gridded items and gravity pointing into the screen is shown.
2. A shake of the device decreases the gravity so that it is possible to rearrange the icons.

Web browser - Press - Changed gravity

When the close button is pressed the gravity is increased so that the web browser collapses and disappears from the screen.

Menu system - Long Tap - Spring

- Menu: Rope ladder

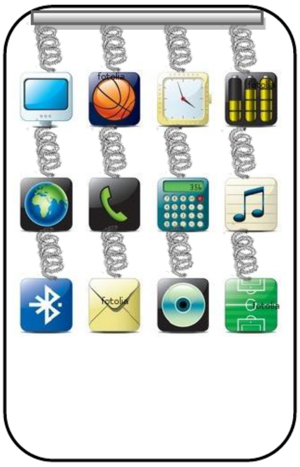

Figure A.9: Rope ladder

- Menu: Tabbed menu

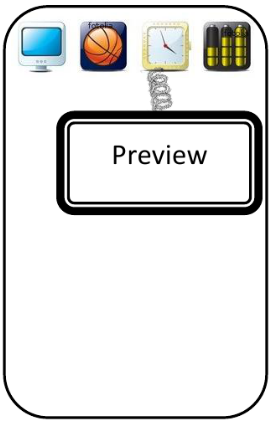

Figure A.10: Tabbed menu

- Menu: Radial madness.

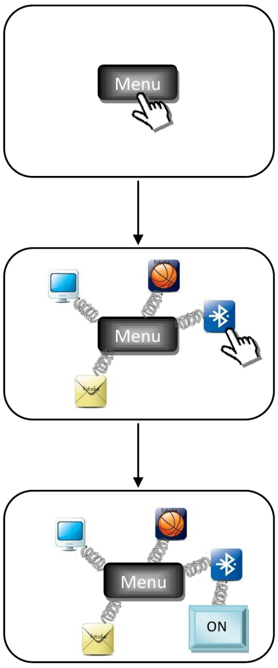

Figure A.11: “Radial Madness”

Screen saver - Drag & Release - Hinge

The screen saver is attached to one side of the screen with a vertical hinge. To unlock the device a drag input opens the screen saver “door”.

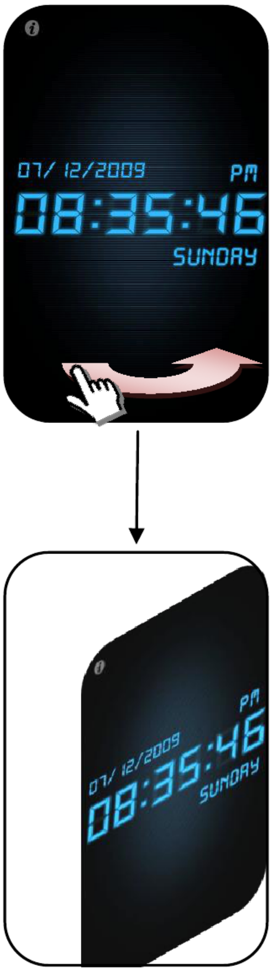

Figure A.12: Screen Saver - Drag & Release - Hinge

Contact List - Shake - Changed mass

1. The contact list is shown as a horizontal bar on the top of the screen.
2. A shake changes the mass of the contacts so that the list unfolds and covers the whole screen.
3. A new shake decreases the mass and folds the list back to its original position.

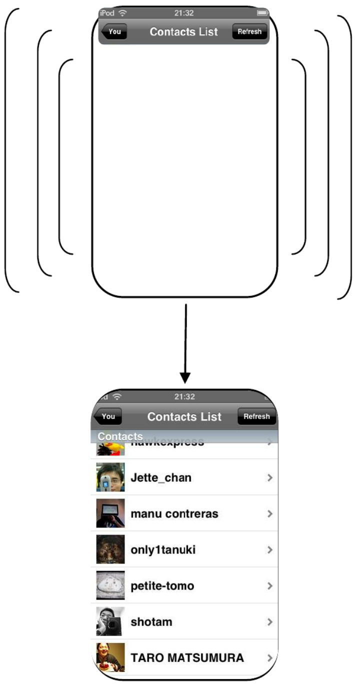

Figure A.13: Contact List - Shake - Changed Mass

Music player - Tilt - Changed Gravity

The device is tilted from a vertical to a horizontal position and the view goes from portrait to landscape. The transition between the two states is like a liquid, i.e. not a change between two discrete states. When the transition is done the music player returns to being a solid object.

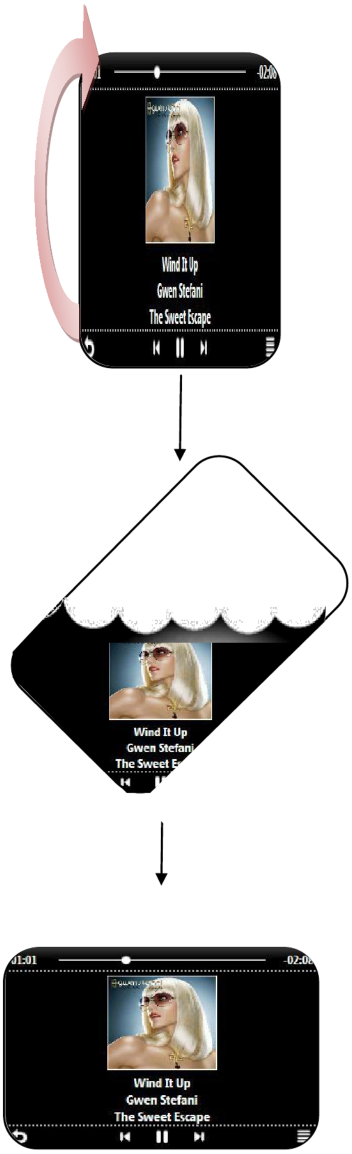

Figure A.14: Music Player - Tilt - Changed Gravity

Music player - Drag & Release - Hinge

Switch between artists/albums/songs by opening and closing "doors" attached with hinges.

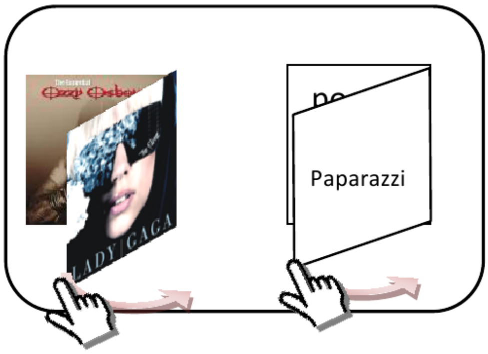

Figure A.15: Music Player - Drag & Release - Hinge

A.2 The free part

General comments

- Physics enabled UIs might be difficult to control. Undesired or unexpected behaviour can occur.
- Randomized actions might be fun, but is it useful?
- Useful as eye candy. E.g. one can use a liquid to represent the battery level in the power meter.

UI scenarios

- Could be preferable in a screen saver where any physical properties can be applied.
- Could be used to deform a list when it reaches the end.
- Could be used as an additional information layer, e.g. when the battery level is low the physical properties of all objects in the UI changes. This could result in heavier and slower actions.
- Music Player where record cases are attached together with a hinge. The cases can be flipped to view the backs of them. Very similar to a cross between the ideas presented in Figure: A.15 and Figure: A.7 on pages 50 and 41.
- A menu system is placed as a bar of icons at the left hand side of the screen. To open for example the contact list the icon is tossed into the opposing wall where it expands to cover the whole screen. See Figure A.16 on page 52.
- The icons are represented as spheres in a menu system organized symmetrically around the centre. To start an application, drag the icon to the centre. It will then expand to cover the screen. See Figure A.17 on page 53.

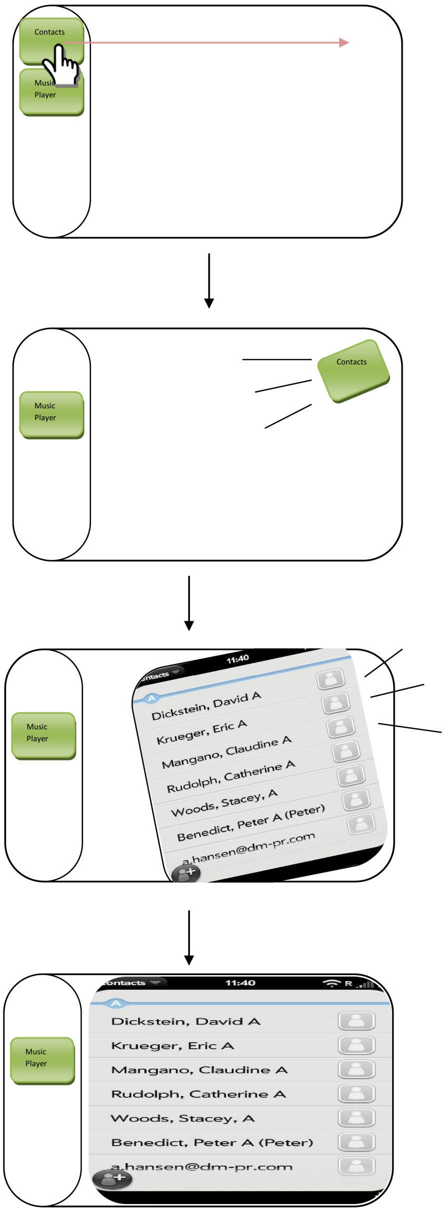

Figure A.16: Figure showing the “Toss into wall” idea from the free part of the workshop

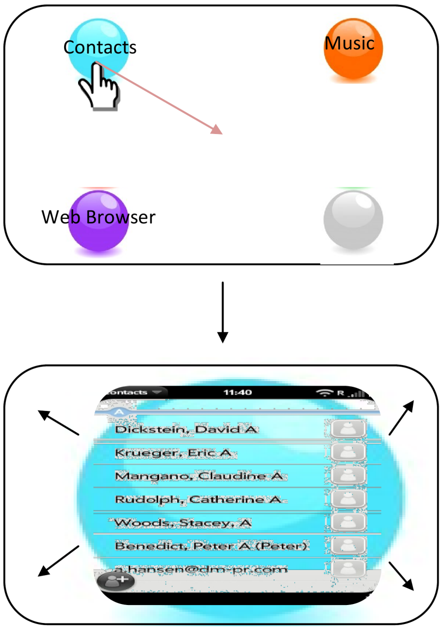

Figure A.17: Figure showing the “Drag to center” idea from the free part of the workshop.

B Appendix: Interview concerning previous work with physics engines on TAT

The interviews was made via e-mail and the question and answers are extracted from a running text to make them more readable.

B.1 Interview with an employee at TAT

- **Question:** In what applications have you tried using a physics engine?
- **Answer:** We used the physics engine in a couple of different applications. One was to make a number of balls bounce around in accordance to input from an accelerometer. Another was when drag-and-dropping 3D meshes. In this case the physics engine was used to attach the meshes to the finger with springs (instead of the meshes being glued to the finger).
- **Question:** In these scenarios what did you find were the gains of using a physics engine?
- **Answer:** In the application with the balls it was perfect to use a physics engine since everything was happening dynamically. In the other scenario it was necessary to use a physics engine to make the meshes swing realistically. It was also necessary in order to make the meshes collide with walls and other meshes when dragged around.
- **Question:** What problems did you encounter?
- **Answer:** First of all it took some work to integrate the physics engine, Bullet Physics 3D (see Section 4.3.1), with Cascades. We also found out that the behaviour of the objects in several situations became hard to control. Often we would have liked to have some restrictions on the objects, which proved to be hard to do in Bullet. E.g. when implementing the drag-and-drop on widgets. They hanged nicely from the finger and collided correctly with the walls and other objects. However, this made it easy to make the widget start spinning so that it ended up up-side-down or in other unwelcome positions, e.g. resting on the wrong edge. We wanted to be able to always see the front of the widget or that it at least would, in an appealing way, make itself end up in the correct position when stabilized. We tried to snap them into place but this turned to look too ugly and unrealistic. Another problem was that there were so many parameters for the objects and constraints in Bullet to tweak, which made it hard and time-consuming to achieve the desired effects in different situations.
- **Question:** Do you have any suggestions for solutions to these problems?
- **Answer:** It would have been good to have a physics engine where you were able to put restrictions, that are not physically correct, on objects in a smooth and easy way. This would remove the use of forced translations and rotations which was the solution we had to use in our applications. E.g. imagine a widget which had started spinning and which had the restriction not to end up up-side-down. Then the physics engine could solve this by perhaps giving the widget an extra impulse in the beginning of the spin to compensate for where it otherwise would have ended up and in that way make sure the object always ended up in the desired position.